- The overridden base method is not a sealed method.

- The override declaration and the overridden base method have the same return type.

- The override declaration and the overridden base method have the same declared accessibility. In other words, an override declaration cannot change the accessibility of the virtual method.

An override declaration can access the overridden base method using a base-access. In the example

```
class A
{
        int x=10;
        public virtual void Display()
        {
                MessageBox.Show("x= {0}",x);
        }

}
class B:A
{
        int y=20;

        public override void Display()
        {
                base.Display();
                MessageBox.Show("y= {0}",y);
        }
}

    private void ButCalculate_Click(object sender, EventArgs e)
    {

                A a = new A();
                B b = new B();
                b.Display();

        }
```

The `base.Display()` invocation in B invokes the Display method declared in A. A base-access disables the virtual invocation mechanism and simply treats the base method as a nonvirtual method. Had the invocation in B been written ((A) this).Display( ), it would recursively invoke the Display method declared in B, not the one declared in A, because Display is virtual and the runtime type of ((A) this) is B.

Only by including an override modifier can a method override another method. In all other cases, a method with the same signature as an inherited method simply hides the inherited method.

In the example

```
class A
{
  public virtual void Display()                              {
                  MessageBox.Show("A.Display");
  }

}
class B:A
{
        public virtual void Display() //Causes a warning.
        {
                MessageBox.Show("B.Display");
        }
}
```

The `Display` method in B does not include an override modifier and therefore does not override the `Display` method in A. Rather, the `Display` method in B hides the method in A, and a warning is reported because the declaration does not include a new modifier.

In the example

```
class A
{

        public virtual void Display()
{
                MessageBox.Show("A.Display");
        }

}
class B:A
{
        new private void Display()
        {
                MessageBox.Show("B.Display");
        }
}
class C:B
{

        public override void Display()
        {
                MessageBox.Show("C.Display");
        }
}

    private void ButCalculate_Click(object sender, EventArgs e)
    {

                A a = new A();
                B b = new B();
                C c = new C();
```